Direct Methods for Solving Sparse Linear Systems of Equations

Esmond G. Ng

(EGNg@lbl.gov)

Lawrence Berkeley National Laboratory

[with input from Xiaoye S. Li (XSLi@lbl.gov, http://www.nersc.gov/~xiaoye)]

Sparse Linear Systems of Equations

- Linear systems arise frequently in large-scale scientific and engineering calculations.
- Examples:
  - Accelerator physics.
  - Chemical process simulations.
  - Device and circuit simulations.
  - Earth and environmental sciences.
  - Fusion energy.
  - Structural analysis.
  - Structural biology.
- The coefficient matrices tend to be large and sparse.
  - Large: common for $n > 100{,}000$.
  - Sparse: most entries are zero.

Gaussian Elimination

- Given a system of linear equations $Ax = b$.
- Consider direct solutions using Gaussian elimination.

- First step of Gaussian elimination:

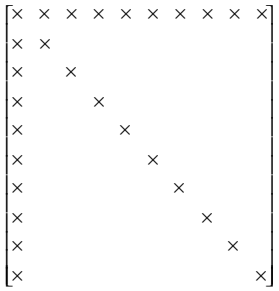
$$A = \begin{bmatrix} \alpha & w^T \\ v & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/\alpha & I \end{bmatrix} \begin{bmatrix} \alpha & w^T \\ 0 & C \end{bmatrix} \quad ; \quad C = B - \frac{vw^T}{\alpha}$$

- Repeat Gaussian elimination on C ...
- Result in a factorization $A = LU$
  - L unit lower triangular, U upper triangular.

- Then x is obtained by solving two triangular systems.

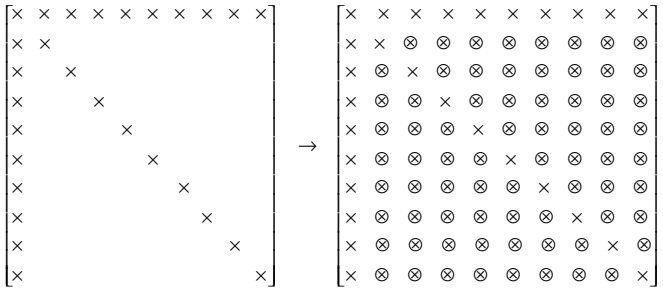
Sparse Gaussian Elimination

- For sparse A:



A has O(n) nonzero entries.

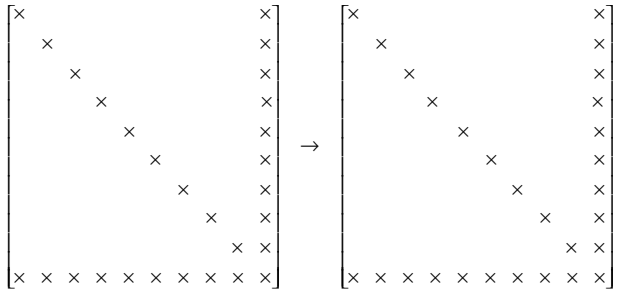
- Gaussian elimination can destroy the zeros.
- Consider $C = B - vw^T/\alpha$.
- Suppose $B_{ij} = 0$. Then $C_{ij} \neq 0$ if v_i and w_j are both nonzero.



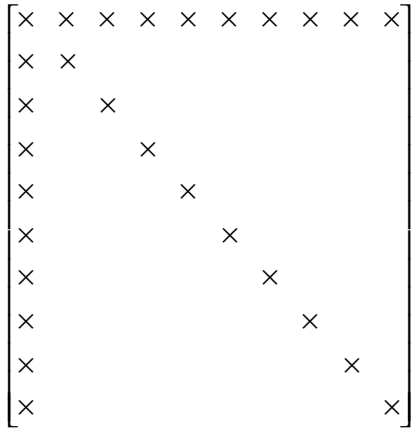
L and U has $O(n^2)$ nonzeros entries.

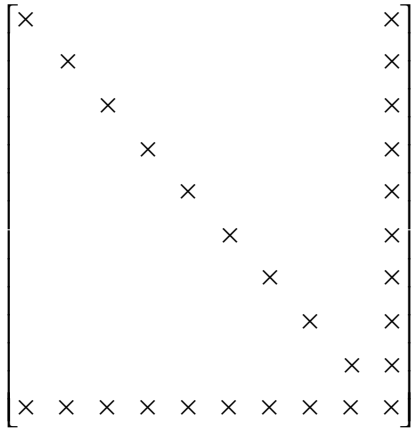
Sparse Gaussian Elimination

- Gaussian elimination can destroy the zero entries.
  - The new nonzero entries are fill entries.

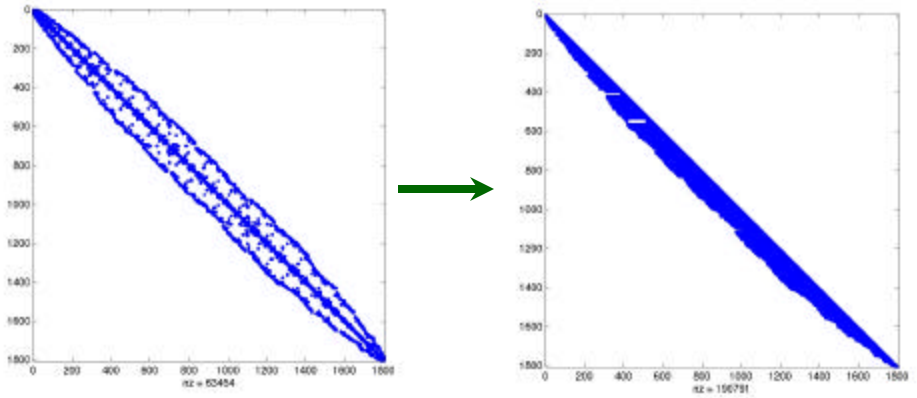


A, L, U have O(n) nonzero entries.

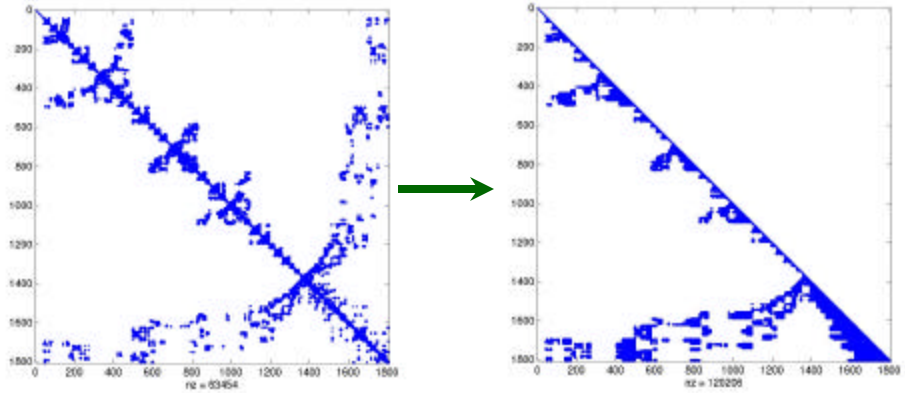


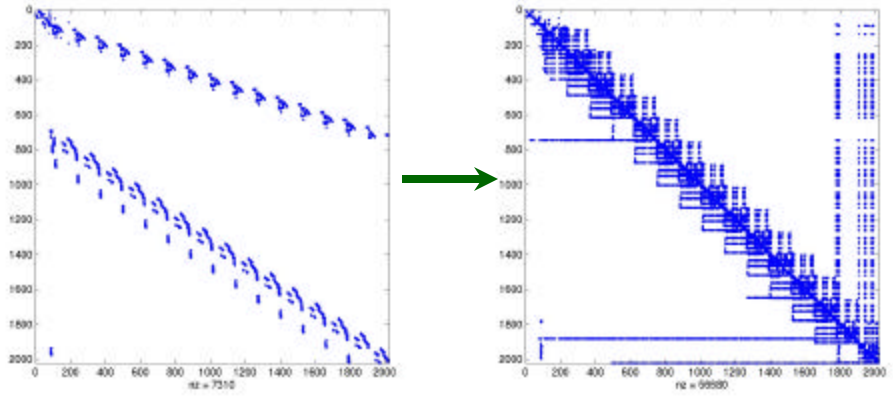


Fill Depends on Sparsity Structure

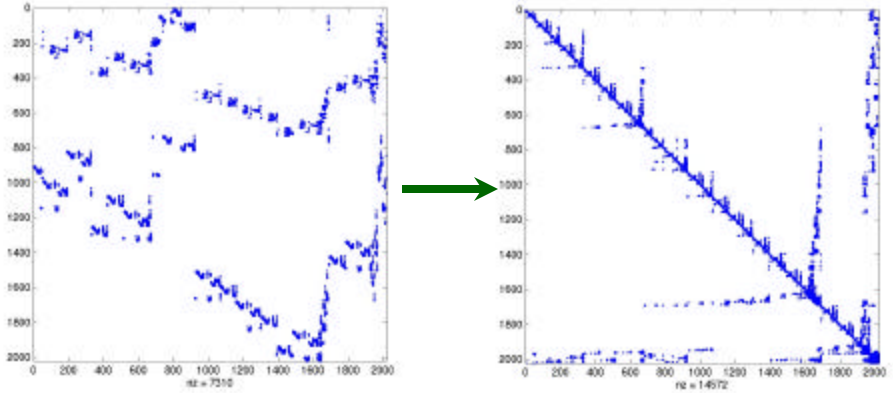


63,454/190,791/120,206





7,310/56,680/14,572



Issues in Sparse Matrix Algorithms

- Control the number of zero entries that turn into nonzero.
  - Sparsity structure of the LU factors depends on sparsity structure of A, which can be changed by permuting its rows/cols.
  - Ordering.
- Predict which zero entries will turn into nonzero.
  - Symbolic factorization.
- Design a storage scheme to store only the nonzero entries.
  - Data structures.
- Find a way to manipulate only the nonzero entries in sparse matrix factorizations.
  - Numerical algorithms.

Dense vs Sparse Gaussian Elimination

- Dense Gaussian elimination: $P_r A P_c = L U$.
  - P_r and P_c are chosen to maintain numerical stability.
  - For partial pivoting, $P_c = I$.

- Sparse Gaussian elimination: $P_r A P_c = L U$.
  - P_r and P_c are chosen to maintain numerical stability and preserve sparsity.

Sparse Gaussian Elimination

- Ingredients in the solution of sparse linear systems:
  - Ordering step:
    - Permute equations and variables to reduce fill in L & U.
  - Symbolic factorization step:
    - Determine locations of nonzeros in L & U.
    - Set up data structures for storing nonzeros of L & U.
    - Allocate memory for the nonzeros.
  - Numerical factorization step:
    - Input numerical values.
    - Compute L & U, with pivoting to maintain numerical stability.
  - Triangular solution step:
    - Use L & U to perform forward and backward substitutions.

Sparse Matrix Factorization

- When numerical stability is not an issue:
  - Ordering, symbolic factorization, and numerical factorization are often distinct and can be performed separately.
  - e.g., sparse symmetric positive definite matrices, diagonally dominant matrices.

- Otherwise:
  - These three phases may have to be interleaved during Gaussian elimination.
  - Why?

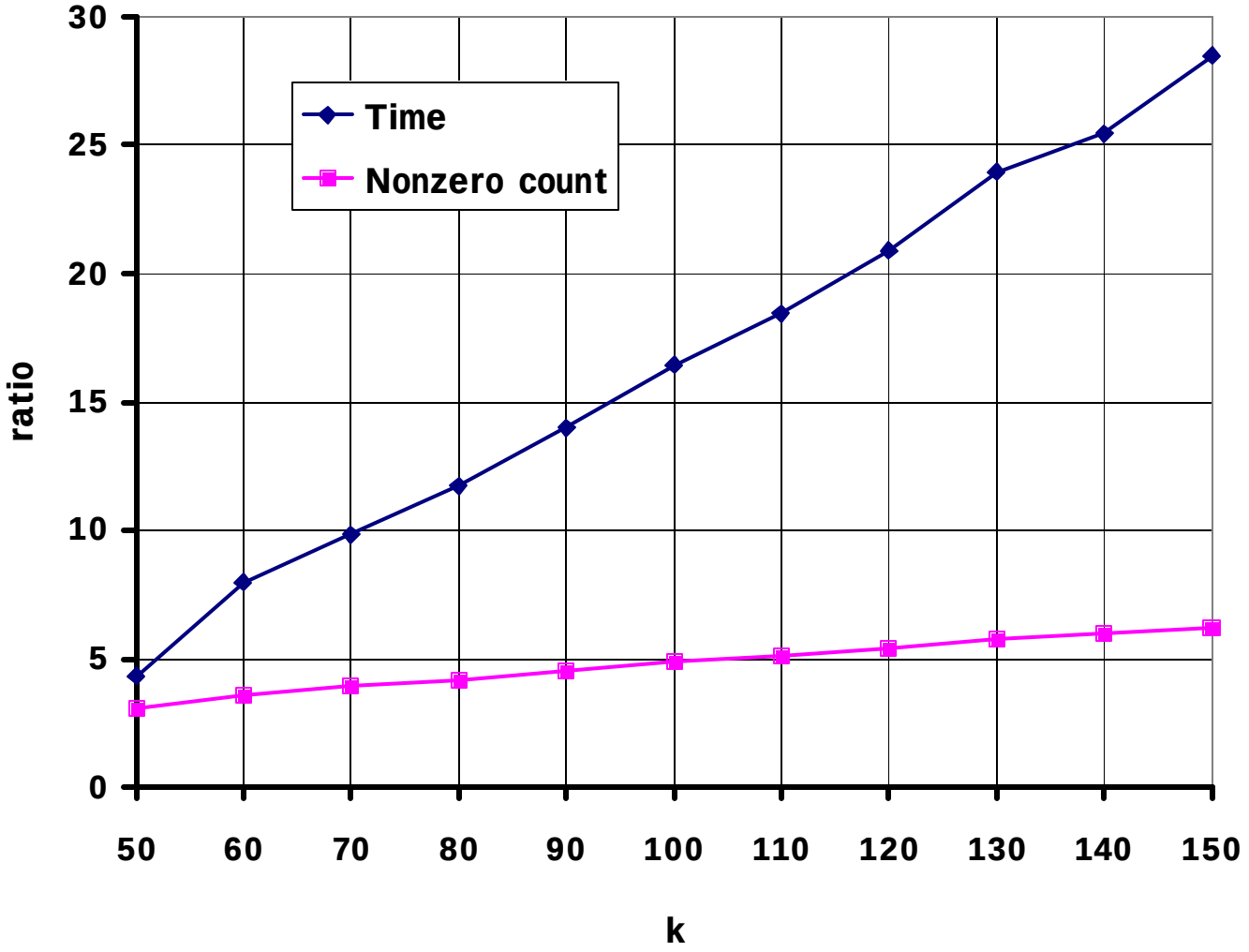
Ordering Algorithms

- Almost all ordering algorithms are combinatorial in nature.
  - O(n!) choices.
- Difficult to find the "best" permutations.
  - An NP-complete problem [Yannakakis '83].

- Almost all ordering algorithms are based on heuristics.
  - Ordering for convenience:
    - Use simple data structures and to design simple factorization algorithms.
      - Banded orderings – put nonzero entries around the diagonal.
  - Ordering for performance:
    - reduce fill as much as possible.
    - Often poised as problems of labeling vertices in graphs.
    - Two classes of fill-reducing heuristics: local vs global.

Banded Orderings vs Fill-reducing Orderings

- Matrices: Defined on k-by-k grids using a 5-point operator.



Ordering Sparse SPD Matrices (Sparse Cholesky)

- Local heuristics: Do the best locally.
  - Minimum degree [Tinney /Walker '67; George/Liu '79; Liu '85; Amestoy/Davis/Duff '94; Ashcraft '95; Duff/Reid '95].
  - Minimum deficiency or fill-in [Tinney/Walker '67; Ng/Raghavan '97; Rothberg/Eisenstat '97].
- Global heuristics: Based on graph partitioning techniques.
  - Nested dissection [George '73; Lipton/Rose/Tarjan '79].
  - Multilevel graph partitioning schemes [Hendrickson/Leland '94; Karypis/Kumar '95].
  - Spectral bisection [Simon et al. '90–'95].
  - Geometric and spectral bisections [Chan/Gilbert/Teng '94].
- Hybrid of the above two.
  - [Ashcraft/Liu '96; Hendrickson/Rothberg '97].

Ordering Sparse Nonsymmetric Matrices

- Symmetric ordering of A^T+A, if no pivoting.

- Symmetric ordering of A^TA, if partial pivoting.
  - Theorem [George/Ng '87]:
    - Suppose the diagonal entries of A are all nonzero.
    - Let $A^TA = R^TR$ and $PA = LU$.
    - Then the sparsity structure of L+U is contained in that of R^T+R, regardless of the choice of P.
  - Making R sparse will make L+U sparse.
    - Find a good symmetric ordering P_c from A^TA.
    - Apply P_c to columns of A [$P_c(A^TA)P_c^T = (AP_c^T)^T(AP_c^T)$].
  - Column orderings based solely on sparsity structure of A.
    - COLMMD in Matlab, COLAMD [Larimore/Davis/Gilbert/Ng '02].

Ordering Sparse Nonsymmetric Matrices

- Nonsymmetric ordering of A.
  - Markowitz scheme [Markowitz '57].
    - Nonsymmetric variant of minimum degree, but apply to all nonzero entries in matrix.
      - cf. complete pivoting for dense matrices.
      - Ignore numerical values – numerical factorization may fail.
  - Markowitz with threshold pivoting [Zlatev '80; Duff/Erisman/Reid '86].
    - Modifying Markowitz scheme by taking numerical values into consideration $\Rightarrow$ usually performed during numeric factorization.
  - Diagonal Markowitz ordering of A, if no pivoting [Amestoy/Li/Ng '02].
    - Same as Markowitz, but apply to diagonal entries only.

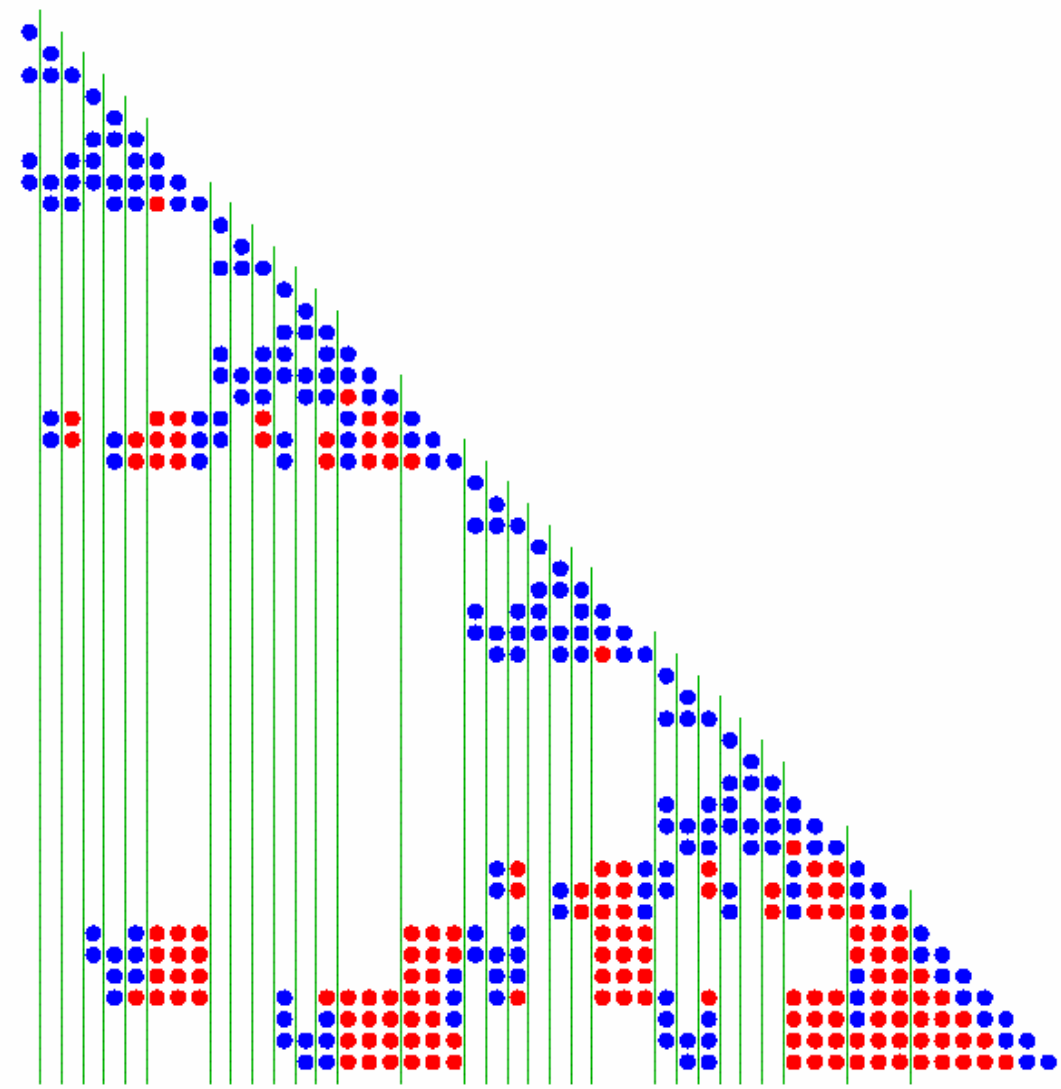
Symbolic Factorization

- Determine the sparsity structure of the factors and set up the data structures for storing the nonzero entries.
  - Determining the sparsity structure?
    - May need to simulate numerical factorization???
      - ⇒ Cost of symbolic factorization is O(flops(LU))???
  - Representing the sparsity structure of the factors?
    - Nonzero entries are usually stored by columns.
    - May need a row index for each nonzero entry???
      - ⇒ Size of the representation is O(|L+U|)???
  - Updating one column by another column?
    - Need index matching or use scatter/gather (indirect addressing)???
      - ⇒ Integer overhead???

Notion of Supernodes

- Consecutive columns with essentially identical sparsity structure can often be found in a triangular factor.
  - A supernode in a triangular factor is a group of consecutive columns { j, j+1, ..., j+t-1 } such that
    - columns j to j+t-1 have a dense diagonal block, and
    - columns j to j+t-1 have identical sparsity pattern below row j+t-1.

Supernodes in Sparse Cholesky Factor



Notion of Supernodes

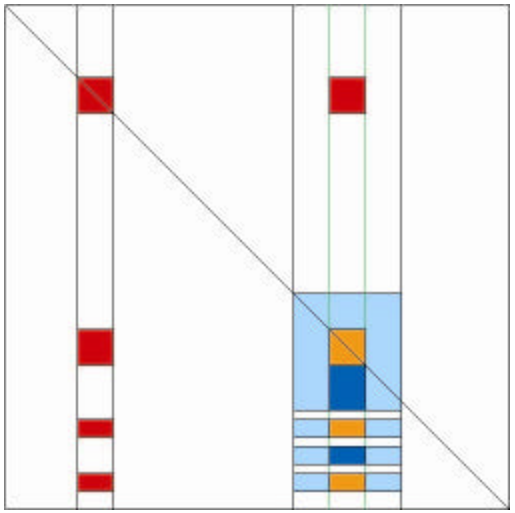
- Consecutive columns with essentially identical sparsity structure can often be found in a triangular factor.
  - A supernode in a triangular factor is a group of consecutive columns { j, j+1, ..., j+t-1 } such that
    - columns j to j+t-1 have a dense diagonal block, and
    - columns j to j+t-1 have identical sparsity pattern below row j+t-1.

- The supernodes provide a partition of the columns of the triangular factor.

- Symmetric case: [Ashcraft et al. '87; Duff/Reid '83; Rothberg/Gupta '91; Ng/Peyton '93].
- Nonsymmetric case: [Eisenstat/Gilbert/Liu '93; Demmel/Eisenstat/Gilbert/Li/Liu '95].

Roles of Supernodes in Factorization

- Let J and K be two distinct supernodes and suppose $k \in K$.
  - Column k is modified by either all columns of J or no columns of J.
  - Multiple columns of K may be modified by all columns of J.



- Benefits:
  - Permit use of level-3 BLAS to take advantage of cache memory.
  - Reduce inefficient indirect addressing (scatter/gather)
  - Allow a compact representation of the sparsity structure of the triangular factor.

- How are the supernodes determined [Liu/Ng/Peyton '93; Gilbert/Ng/Peyton '94]?

Back to Symbolic Factorization

- Symmetric positive definite matrices:
  - Complexity is linear in the size of the representation [George/Liu '80].
  - The size of the representation is less than O(|L|).

- Nonsymmetric matrices with partial pivoting:
  - Complexity cannot exceed O(flops(LU)), but more than O(|L+U|) [Gilbert/Periels '88; Eisenstat/Liu '92/'93; Gilbert/Liu '93].
  - The size of the representation is less than O(|L+U|).

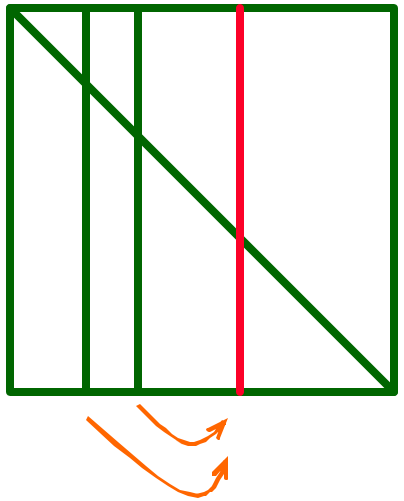
Numerical Factorization

- The goal of general sparse matrix factorization algorithms is to manipulate and operate on nonzero entries only.
  - Unlike dense matrix factorization, integer computation can be non-trivial.
- They can be "left-looking" or "right-looking", and are sparse analogues of dense algorithms.
- New-generation sparse matrix factorizations exploit the supernodal structure and use level-3 BLAS operations as much as possible (à la LAPACK).
- Recent developments focus on
  - Superscalar processor with hierarchical memory.
  - Parallelism.

Left-looking vs Right-looking

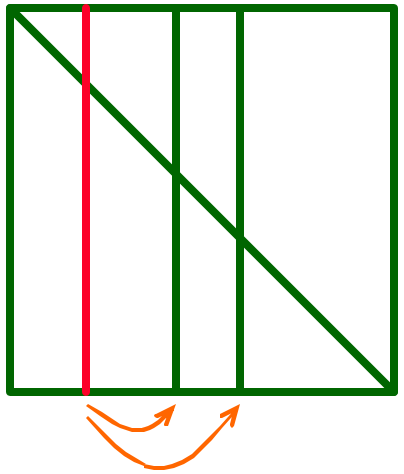
- Left-looking algorithms

```
for j = 1 to n do
  for each k such that Ukj ≠ 0 do
    cmod(j,k)
  end for
  cdiv(j)
end for
```



- Right-looking algorithms

```
for j = 1 to n do
  cdiv(j)
  for each k such that Ujk ≠ 0 do
    cmod(k,j)
  end for
end for
```



Left-looking vs Right-looking

- Many efficient left-looking sparse matrix factorizations.
  - BlkFct [Ng/Peyton '90] for sparse Cholesky.
  - SuperLU [Demmel/Eisenstat/Gilbert/Li/Liu '95] for sparse LU.
- Right-looking algorithms can be inefficient.
  - MA28 [Duff '77], MA48 [Duff/Reid '96] for sparse LU.
- Multifrontal methods – an efficient compromise.
  - MA27 [Duff/Reid '82], MA47 [Duff/Reid '93] for sparse Cholesky (and symmetric indefinite).
  - UMFPACK [Davis/Duff '97/'99] for sparse LU.
- Many implementations for distributed-memory parallel computers are right-looking.

Performance of Sparse Cholesky Factorization

- Matrices: Defined on k-by-k grids using a 5-point operator.
- Left-looking sparse supernodal Cholesky factorization.

k	n	|A|	Ordering Time	SF Time	NF Time	Soln Time	Number of Supernodes	Number of Indices	|L|
50	2,500	7,400	0.02	0.01	0.06	0.04	1,892	15,145	35,943
60	3,600	10,680	0.02	0.02	0.07	0.06	2,720	22,007	54,215
70	4,900	14,560	0.03	0.01	0.11	0.08	3,697	30,376	79,524
80	6,400	19,040	0.04	0.02	0.16	0.11	4,825	39,996	110,462
90	8,100	24,120	0.06	0.03	0.21	0.14	6,102	51,057	145,945
100	10,000	29,800	0.06	0.05	0.27	0.17	7,530	63,209	185,673
110	12,100	36,080	0.08	0.05	0.35	0.21	9,107	77,142	238,524
120	14,400	42,960	0.11	0.06	0.42	0.25	10,835	91,965	289,179
130	16,900	50,440	0.11	0.08	0.51	0.29	12,712	108,411	346,500
140	19,600	58,520	0.13	0.09	0.63	0.34	14,740	126,022	419,198
150	22,500	67,200	0.15	0.10	0.78	0.40	16,917	145,349	496,538

Observations

- Symbolic factorization takes very little time.
  - Little (integer) computing required.
  - Hard to improve by parallelization.
  - Only reason to parallelize is because of problem size or matrix distribution.
  - Parallel implementations have been made for symmetric matrices; speedups are fair.
    - [George/Heath/Liu/Ng '87; Zmijewski/Gilbert '88; Gilbert/Hafsteinsson '90; Ng '93].
- Ordering can be relatively inexpensive too.
  - Local heuristics are very hard to parallelize.
  - Global heuristics based on graph partitioning – divide-and-conquer type approaches – can be parallelized.
    - ParMETIS [Karypis/Kumar '96/'97; Schloegel/Karypis/Kumar '97/'00].

Observations

- Numerical factorization and triangular solution are the most time-consuming.
  - Almost all effort on parallelizing sparse matrix factorization has focused on numerical factorization.
  - Amestoy, Ashcraft, Demmel, Duff, George, Gilbert, Gupta, Heath, Li, Ng, Raghavan, Rothberg, Yang, …
- Will discuss parallel sparse triangular solution next time …
- Focus on sparse numerical factorization.

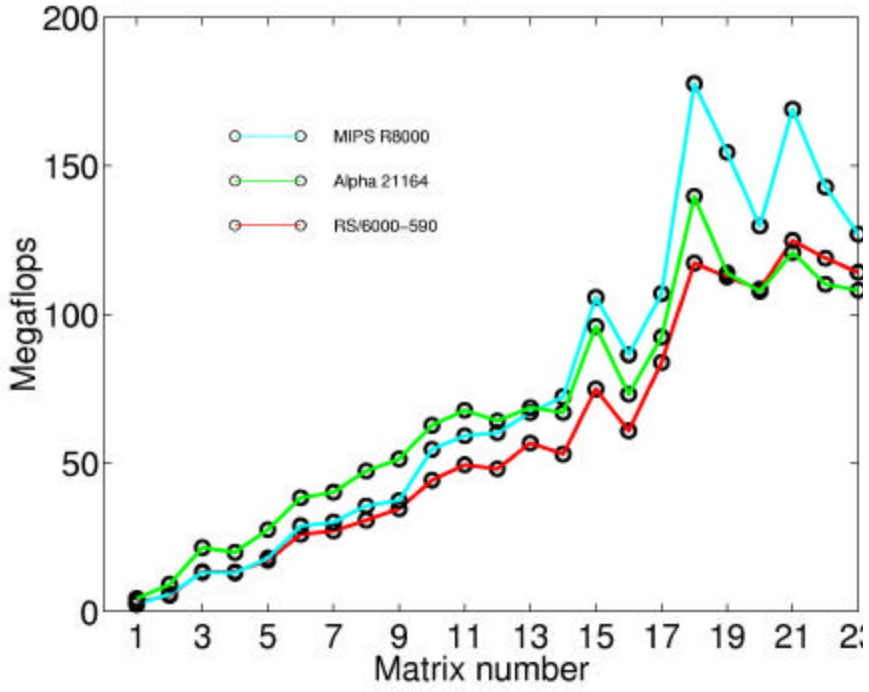
Performance of Serial SuperLU

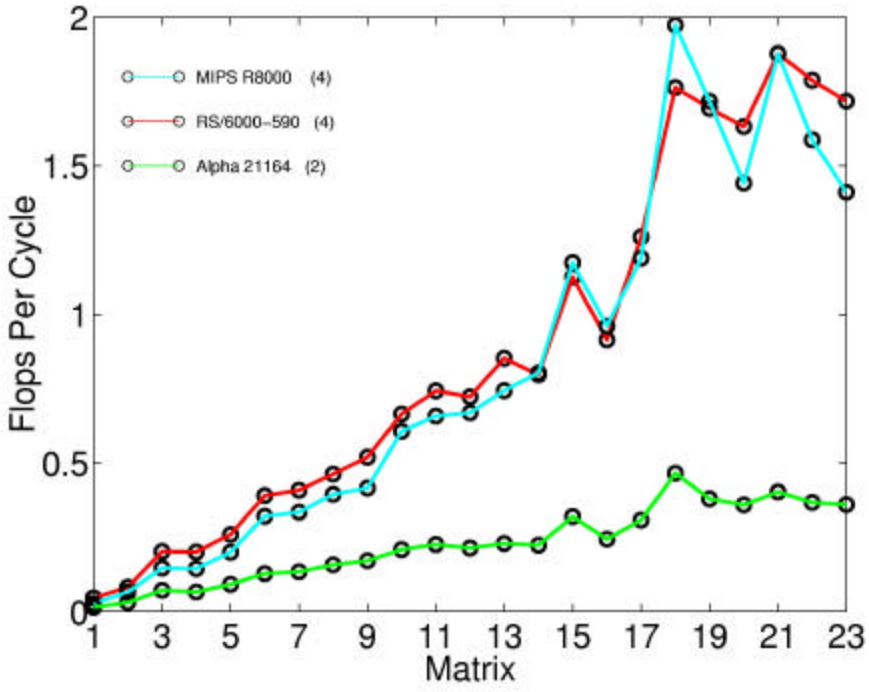
Test Matrices (ordered using COLMMD)

	Name	n	|A|	|A|/n	|L+U|/|A|	#flops (10^6)	#flops/|L+U|
1	Memplus	17,758	99,147	5.6	1.4	1.8	12.5
2	Gemat11	4,929	33,108	6.7	2.8	1.5	16.3
3	Rdist1	4,134	94,408	22.8	3.6	12.9	38.1
4	Orani678	2,529	90,158	35.6	3.1	14.9	53.3
5	Mcfe	765	24,382	31.8	2.8	4.1	59.9
6	Lnsp3937	3,937	25,407	6.5	16.8	38.9	91.1
7	Lns3937	3,937	25,407	6.5	17.7	44.8	99.7
8	Sherman5	3,312	20,793	6.3	12.0	25.2	101.3
9	Jpwh991	991	6,027	6.1	23.4	18.0	127.7
10	Sherman3	5,005	20,033	4.0	21.6	60.6	139.8
11	Orsreg1	2,205	14,133	6.4	28.5	59.8	148.6
12	Saylr4	3,564	22,316	6.3	29.3	104.8	160.0
13	Shyy161	76,480	329,762	4.3	23.2	1,571.6	205.8
14	Goodwin	7,320	324,772	44.4	9.6	665.1	213.9
15	Venkat01	62,424	1,717,792	27.5	7.6	3,219.9	247.9
16	Inaccura	16,146	1,015,156	62.9	9.8	4,118.7	414.3
17	Af23560	23,560	460,598	19.6	30.4	6,363.7	454.9
18	Dense1000	1,000	1,000,000	1,000.0	1.0	666.2	666.2
19	Raefsky3	21,200	1,488,768	70.2	11.8	12,118.7	690.7
20	Ex11	16,614	1,096,948	66.0	23.8	26,814.5	1,023.1
21	Wang3	26,064	177,168	6.8	74.9	14,557.5	1,095.5
22	Raefsky4	19,779	1,316,789	66.6	20.3	31,283.4	1,172.6
23	Vavasis3	41,029	1,683,902	41.0	29.2	89,209.3	1,813.5

Performance of Serial SuperLU

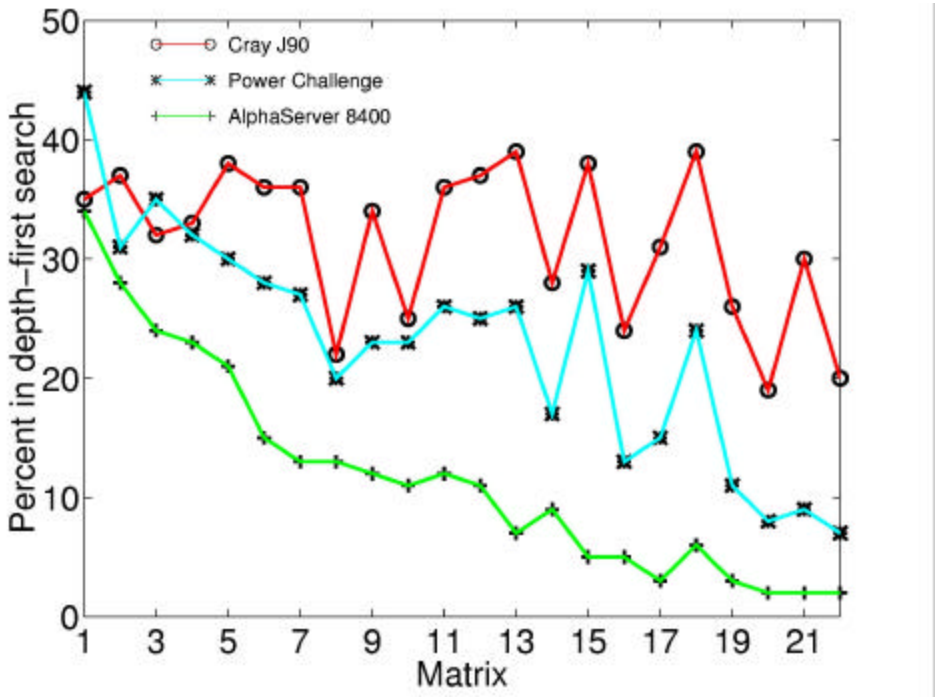
- Time includes everything except column ordering.





Performance of Serial SuperLU

- Show fraction of time in symbolic factorization.
- Show relative strength of integer vs. floating-point speed.
- Roughly carry over to shared memory parallel code.



Parallel Dense Numerical Factorization

- Consider a left-looking implementation:

```
for j = 1 to n do
  for k = 1 to j-1 do
    cmod(j,k)
  end for
  cdiv(j)
end for
```

- Suppose that columns j_1 and j_2, with $j_1 \neq j_2$, are assigned to different processors.
- If column k of the factors, with $k < j_1, j_2$, are made available to the processors containing columns j_1 and j_2, cmod(j_1,k) and cmod(j_2,k) can be performed in parallel.
- The cdiv's have to be completed sequentially.

Parallel Sparse Numerical Factorization

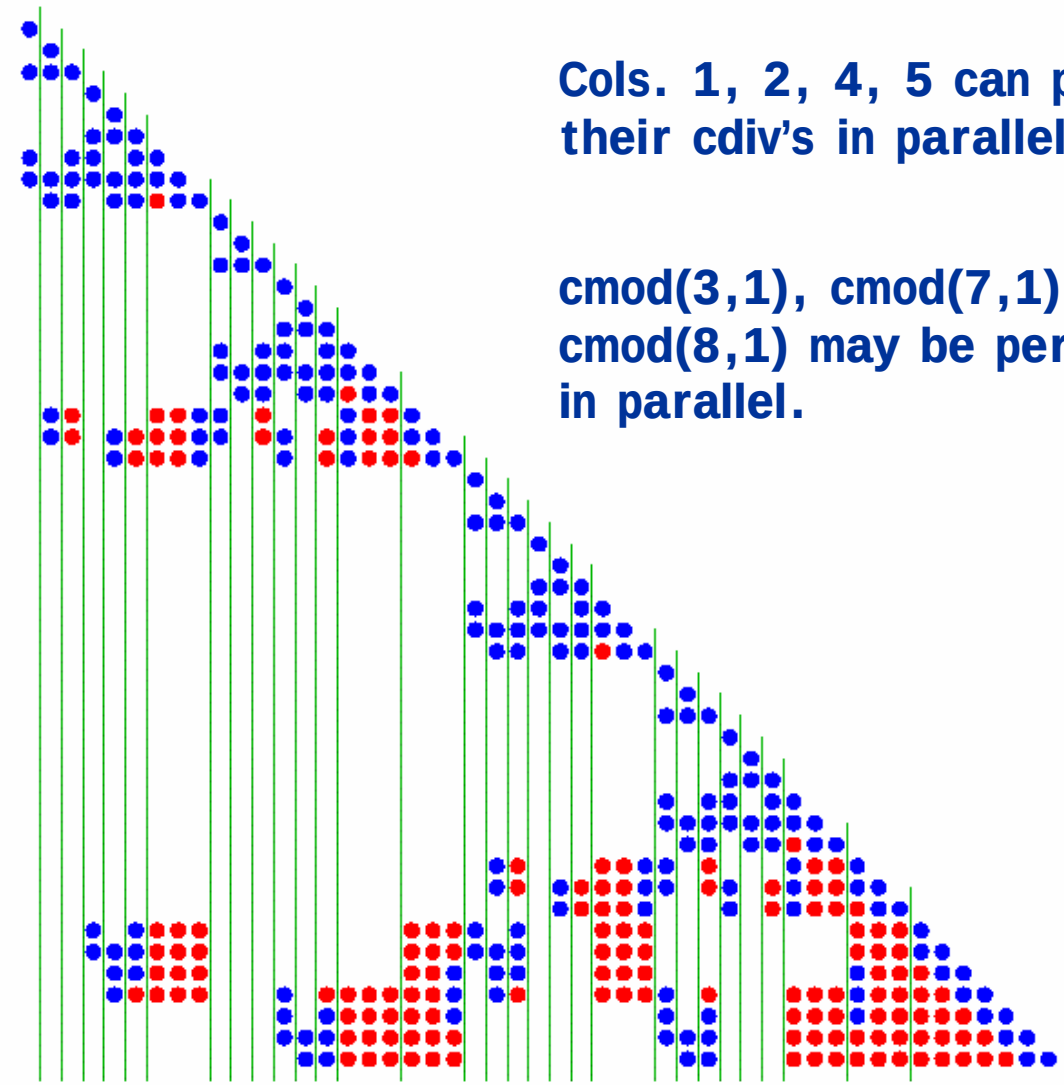
- Sparse left-looking factorization:

```
for j = 1 to n do
  for each k such that U_kj ≠ 0 do
    cmod(j,k)
  end for
  cdiv(j)
end for
```

- As in the dense case, many of the cmod operations can be performed in parallel.

- Because of sparsity, some of the cdiv's operations can also be performed in parallel.
  - Potentially higher degree of parallelism.

Parallel Sparse Cholesky Factorization

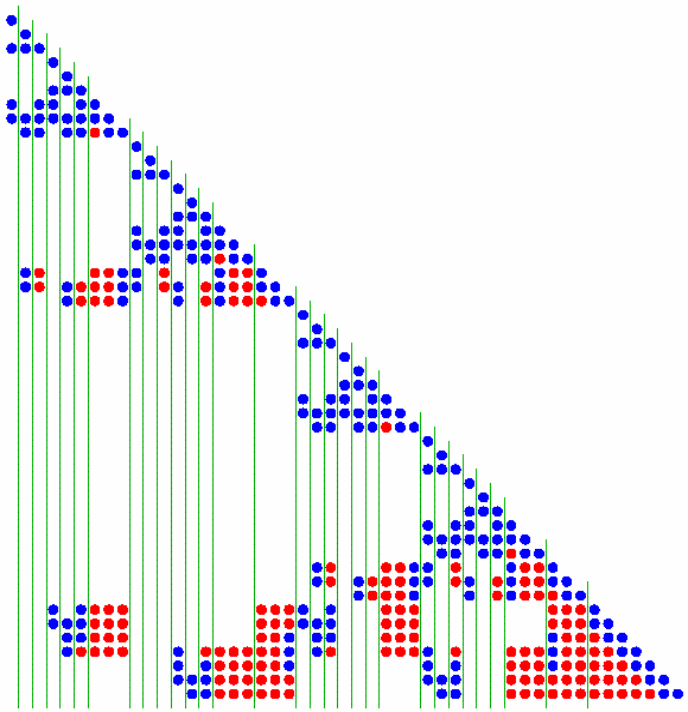


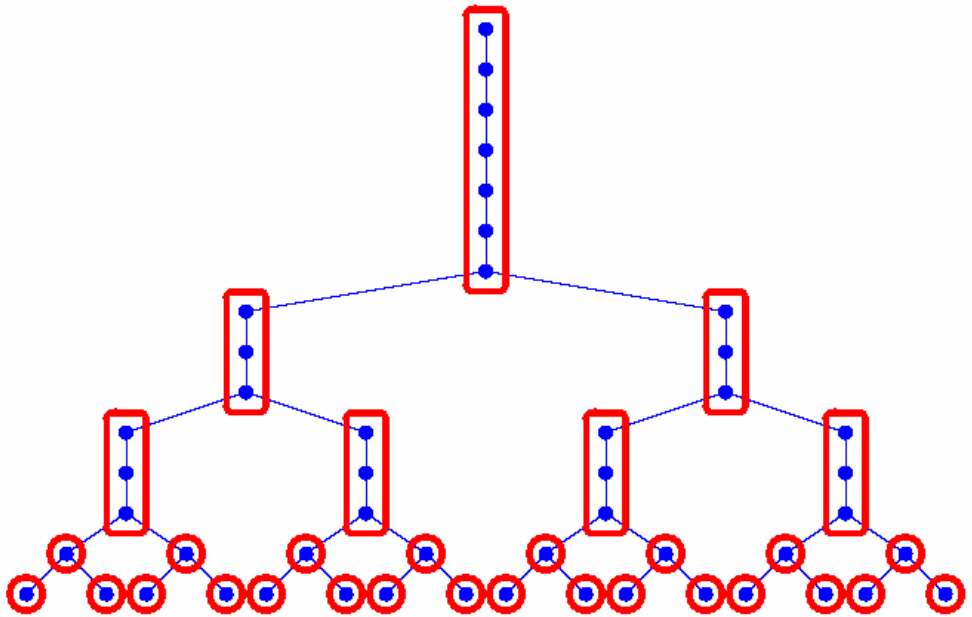
Cols. 1, 2, 4, 5 can perform their cdiv's in parallel.

cmod(3,1), cmod(7,1), and cmod(8,1) may be performed in parallel.

Identifying Parallel Tasks

- For sparse symmetric positive definite matrices, elimination tree is a powerful and useful tool.
  - The elimination tree of a sparse Cholesky is an acyclic graph defined in terms of the first off-diagonal nonzero entries in the columns of the Cholesky factor [Schreiber '82; Liu '90].

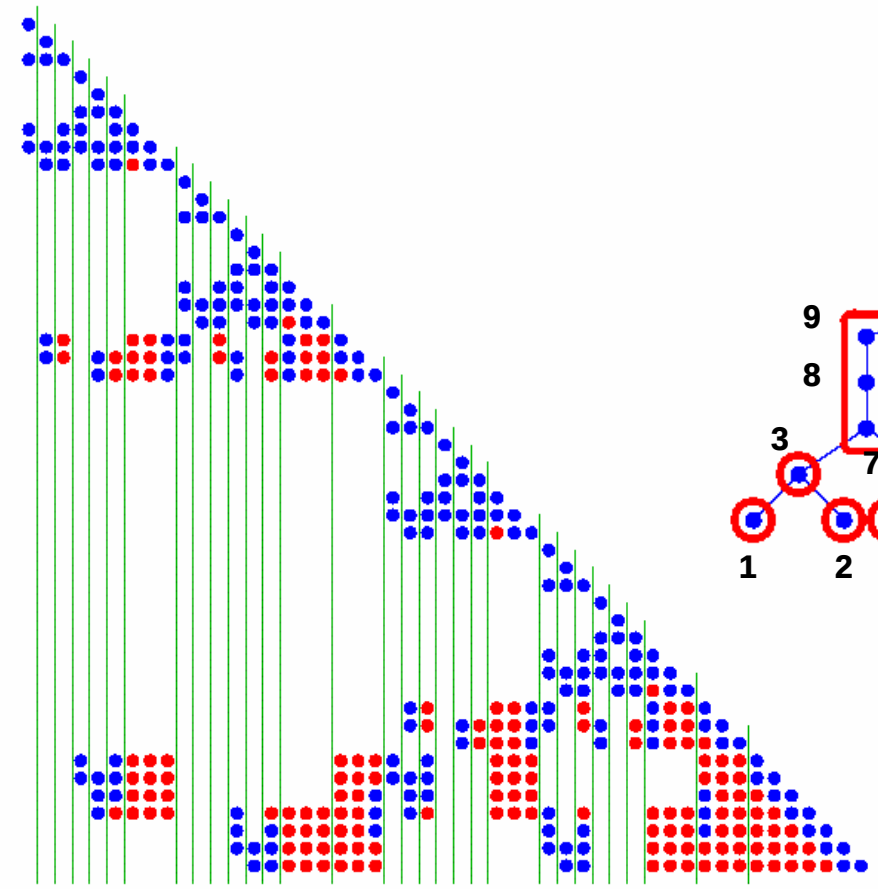


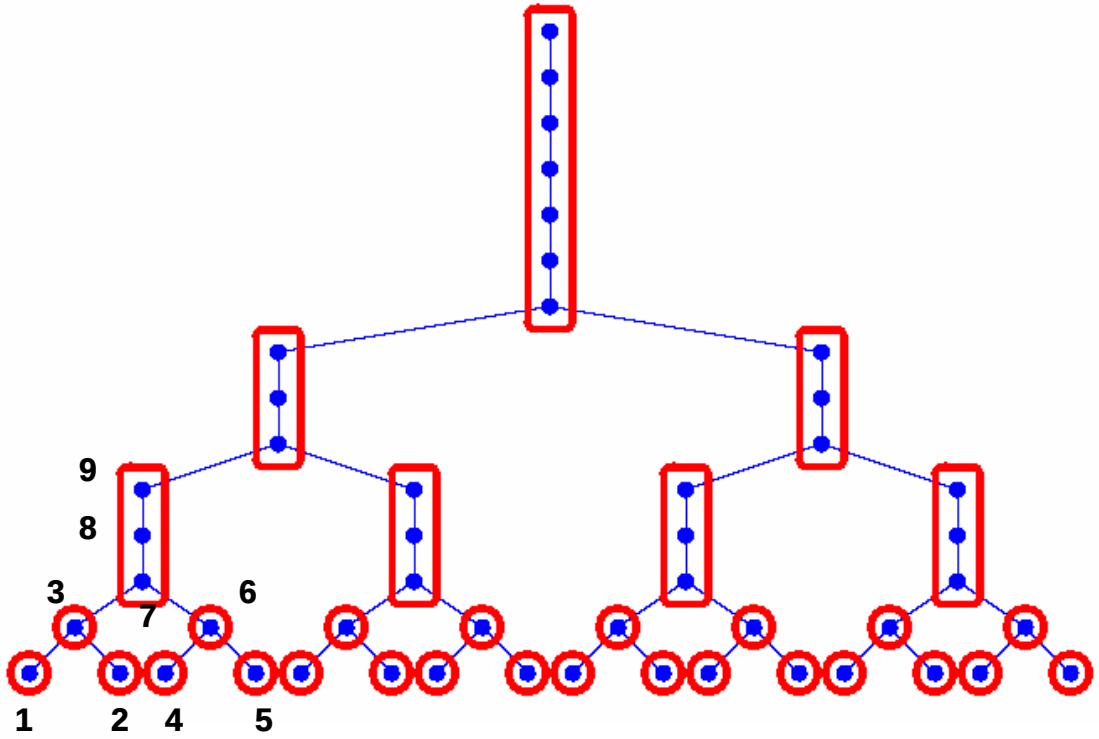


Identifying Parallel Tasks

- Each vertex in the elimination tree corresponds to a column in the Cholesky factor.
- The elimination tree can be used to characterize many properties of sparse Cholesky factorization, as well as the sparsity structure of the Cholesky factor.
  - All the leaves of the elimination tree correspond to columns whose cdiv's are independent and can be performed simultaneously.
  - Suppose that cdiv(j) has been performed. Then there is a cmod(k,j) if vertex k belongs to the path between vertex j and the root of the elimination tree.
  - Disjoint subtrees represent independent subsets of columns that can be computed simultaneously.

The Elimination Tree





Parallel Sparse Cholesky Factorization

- The elimination tree can be used to identify parallel tasks.
- It can used to assign/schedule work among the processors.
- It can be used to study the load balancing issue.
- How is the elimination tree computed?
  - The complexity of computing the elimination tree from the sparsity structure of A is $O(|A|\ \alpha(n,|A|))$ [Liu '86].
- The supernodes can be determined once the elimination tree is known.

Parallel Sparse LU Factorization

- For sparse nonsymmetric matrices (with partial pivoting), the proper tool to use is a directed acylic graph (DAG) [Gilbert/Liu '93], which has been simplified by the use of symmetric pruning [Eisenstat/Liu '92].
- The DAG is not known until the numerical factorization is performed.
- Instead, use the elimination tree of $A^T A$ (also called column elimination tree of A) [Gilbert/Ng '93].
  - Served as an upper bound.
  - Can update the tree on the fly as the numerical factorization proceeds [Demmel/Gilbert/Li '97].

Shared-memory Sparse Matrix Factorization

- Relatively easy to implement.

- Use the elimination tree to guide the assignment of work among the processors.
- Tasks are usually scheduled dynamically.
  - Gradually removing the “leaves” level by level.
- Load balancing is easy to achieve.

- Relative good performance when number of processors is small and the cost of synchronization is low.
- [George/Heath/Liu/Ng ’86; Ng/Peyton ’93; Rothberg/Gupta/Ng/Peyton ’92; Demmel/Gilbert/Li ’99].

Distributed-Memory Sparse Matrix Factorization

- Previous approach becomes ineffective on distributed-memory architectures.
  - The cost of synchronization using message passing becomes expensive.
- Some algorithmic changes are necessary for distributed-memory implementation.
  - 2-D block cyclic mapping.
    - Parallelize the loops over both rows and columns.
  - In the case of sparse LU, switch to static pivoting (GESP)
    - Pivot before numerical factorization so that static data structures can be used.
    - Accommodate possible pivot growth during factorization without changing data structures [George/Ng '87].
    - Decouple symbolic and numerical phases.

Distributed-Memory Sparse Matrix Factorization

- Reducing communication cost is important.
  - [George/Liu/Ng '89; Geist/Ng '89; Raghavan '91].
- Use static scheduling instead of dynamic scheduling.
  - Assign "subtrees" to processors, instead of single columns.
  - Multiple processors can cooperate to compute columns in upper part of the elimination tree.

Performance of SuperLU_MT – Factorization Speed

- Time, Mflops, and speedup on Origin 2000.

Matrix	n	|A|	p=1	p=8	Speedup	p=18	Speedup
Ex11	16,614	1,096,948	209	33	6.3	20	10.5
Raefsky4	19,779	1,316,789	229	39	5.9	25	9.2
Bbmat	38,744	1,771,722	605	166	3.6	64	9.5
Vavasis3	41,092	1,683,902	598	136	4.4	75	8.0
Twotone	120,750	1,224,224	313	58	5.4	38	8.2

- A 3-D flow calculation (EX11).

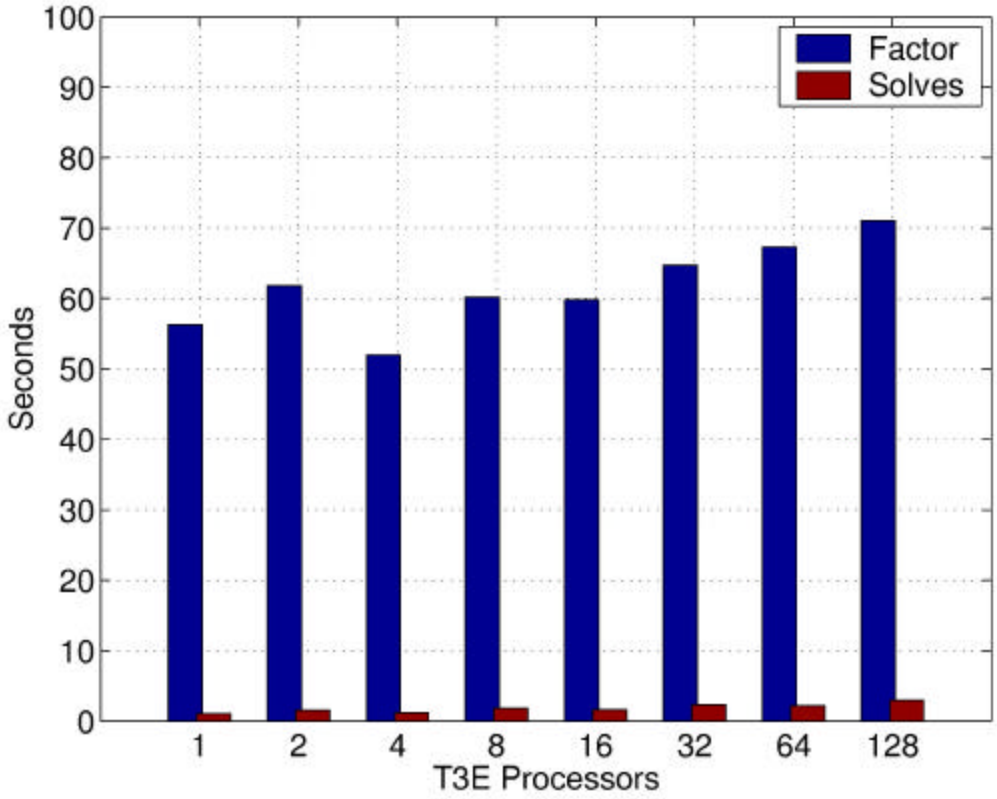
Machine	CPUs	Speedup	Mflops	Percent Peak
C90	8	6	2,583	33%
J90	16	12	831	25%
Power Challenge	12	7	1,002	23%
Origin 2000	20	10	1,335	17%
AlphaServer 8400	8	7	781	17%

Performance of SuperLU_MT – Factorization Speed

Cray C90								
Matrix	|L+U| (10^6)	Flops (10^9)	p=1	p=4	p=8	Speedup	Mflops	Percent Peak
AF23560	14	6.4	36.2	9.4	6.2	4.9	1,035	13%
EX11	26.2	26.8	75.4	21.2	10.4	6.5	2,538	33%
RAEFSKY4	26.7	31.3	78.3	21	13.1	5.5	2,399	31%
AlphaServer 8400								
Matrix	|L+U| (10^6)	Flops (10^9)	p=1	p=4	p=8	Speedup	Mflops	Percent Peak
AF23560	14	6.4	70.8	18.1	11.6	5.8	553	12%
EX11	26.2	26.8	245.3	64.6	34.2	7.1	781	16%
RAEFSKY4	26.7	31.3	288.2	74.1	42.8	6.6	734	15%
Origin 2000								
Matrix	|L+U| (10^6)	Flops (10^9)	p=1	p=8	p=18	Speedup	Mflops	Percent Peak
EX11	26.2	26.8	209.6	33.1	20.3	10	1,335	19%
RAEFSKY4	26.7	31.3	229.5	39.2	25.7	9	1,222	17%
BBMAT	50.4	45.5	605.3	166.3	64.1	9	710	10%
TWOTNE	23.9	12.5	313.4	57.9	39.3	8	318	5%

SuperLU_Dist – Scaling on T3E

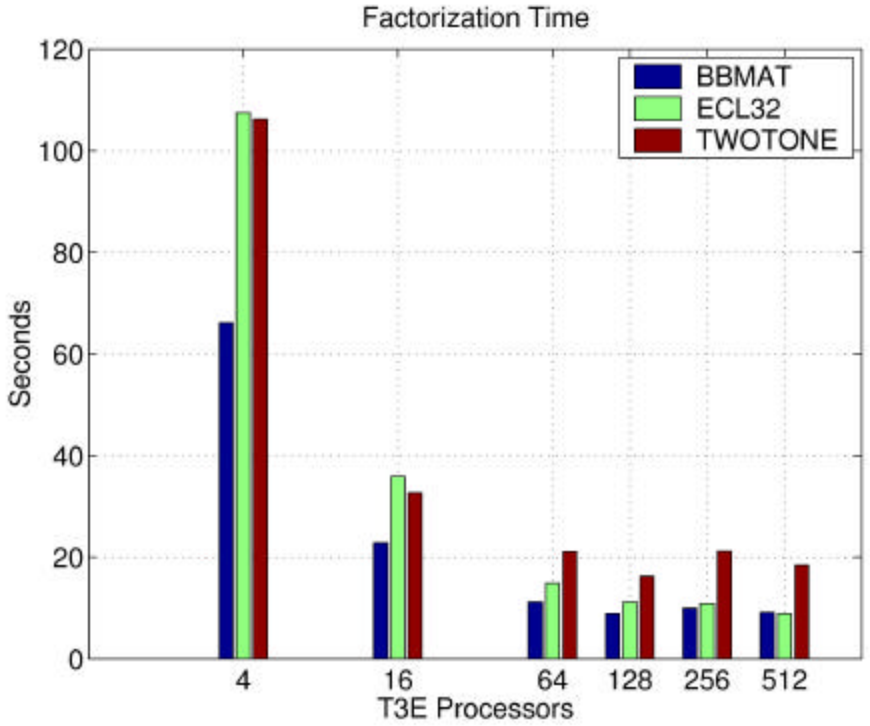
- 3D grids, 11-point stencil.
- Grid size increases with number of processors, such that flops per processor is roughly constant.

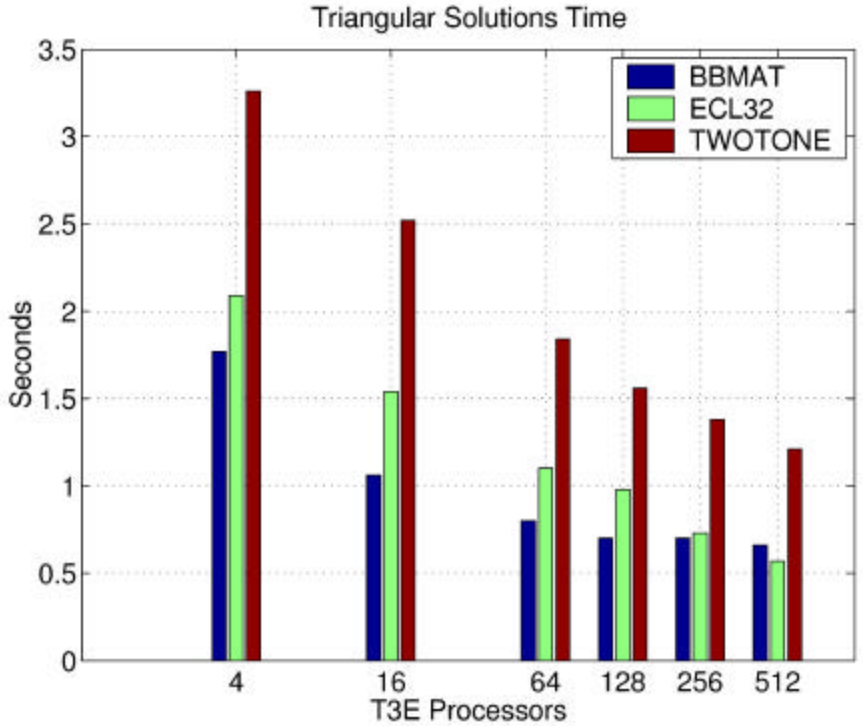


SuperLU_Dist on Irregular Problems

- Minimum degree ordering on A^T+A.

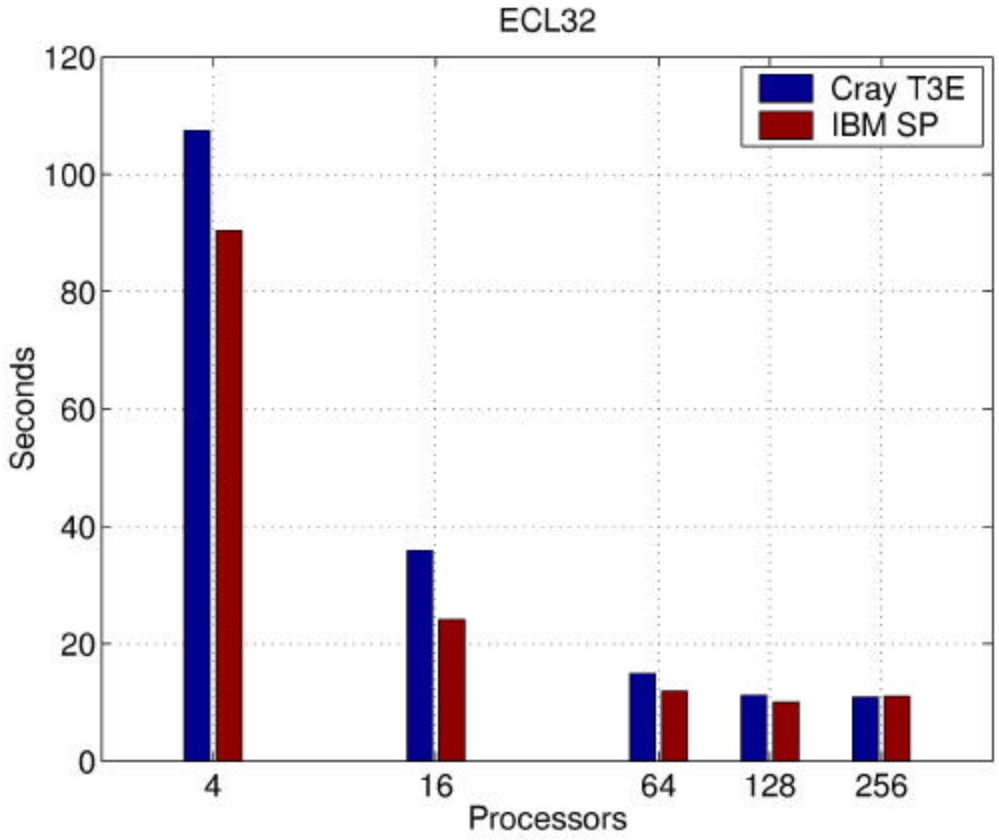
Matrix	Discipline	Symmetry	n	|A|	|L+U| (10^6)	Flops (10^9)
BBMAT	CFD	0.54	38,744	1,771,722	35.9	26.6
ECL32	Device Simulation	0.93	51,993	380,415	41.4	60.6
TWOTONE	Circuit Simulation	0.43	120,750	1,224,224	10.7	7.3





SuperLU_Dist on Irregular Problems

- Comparing time on Cray T3E and IBM SP.



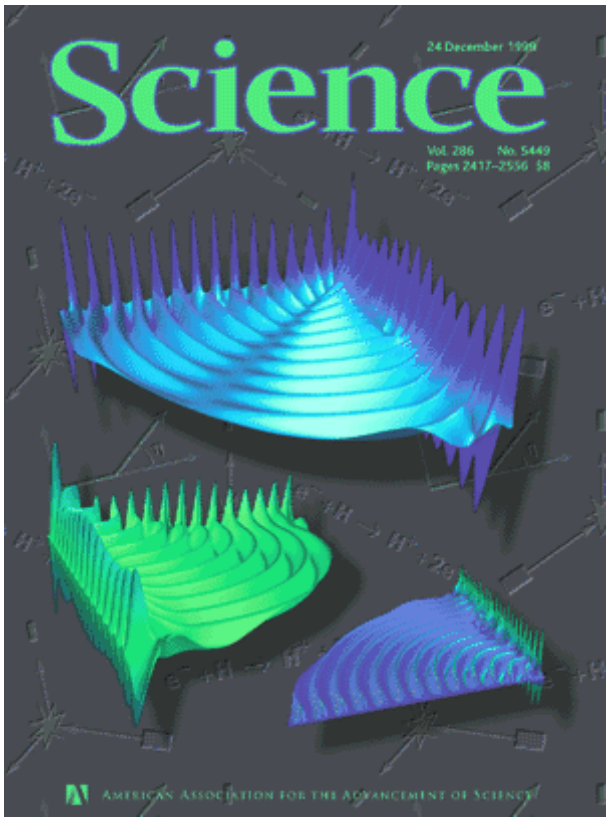
SuperLU_Dist – Impact of Ordering on Scalability

- Matrix ECL32 – Time on Cray T3E.

Ordering	|L+U| (10^6)	p=4	p=32	p=128	p=512	Gflops/s
MMD(A'A)	73.5	325.0	60.5	21.5	14.3	8.5
MMD(A'+A)	41.4	107.4	20.6	11.1	8.9	6.8
ND(A'+A)	24.3	49.0	12.0	8.7	9.6	2.2

Application of Sparse Gaussian Elimination

- First solution to quantum scattering of 3 charged particles.
  - [Rescigno/Baertschy/Isaacs/McCurdy, Science, Dec 24, '99].
- Need to factor large sparse complex nonsymmetric matrices.
  - n = 209,764; |A| = 1,046,988.
    - Factor nonzero count: 12,838,222.
    - Factor time: 2 minutes on 16 processors of LBNL's Cray T3E.
  - n = 1,792,921; |A| = 8,959,249.
    - Factor nonzero count: 143,643,265.
    - Factor time: 48 minutes on 24 processors of LLNL's IBM SP (ASCI Blue-Pacific).



Summary

- Very brief overview of sparse direct methods.

- Lots of open issues …
  - Parallel ordering algorithms.
  - Parallel symbolic factorization algorithms.
  - Scheduling and load balancing.
  - Complexity of ordering.
  - Numerical algorithms are converging???
  - …